Istruzioni Macchina

Corso di Architettura degli elaboratori e laboratorio – Modulo Laboratorio

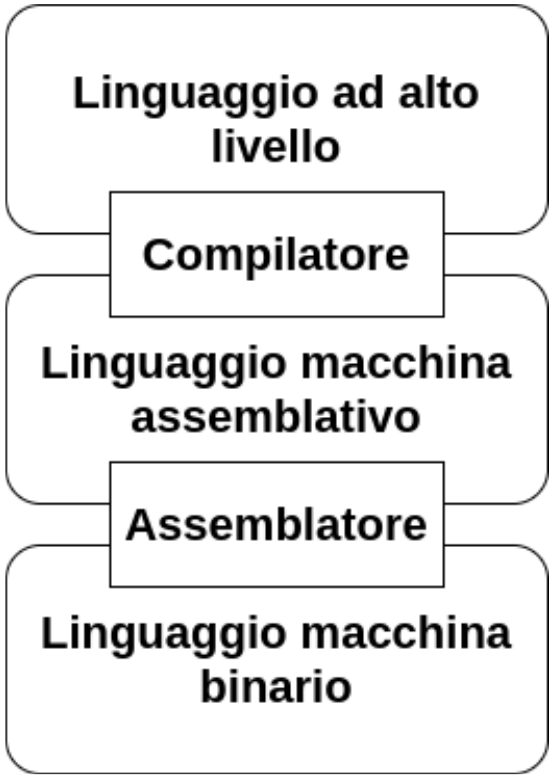
Gabriella Verga

Come funziona?

- Il calcolatore elettronico esegue SEQUENZIALMENTE una serie di ISTRUZIONI.
- Le istruzioni definiscono delle operazioni da eseguire e sono raggruppate in PROGRAMMI.
- Spesso le operazioni devono essere eseguite su dei DATI.
- L'utente può interagire con il calcolatore tramite le INTERFACCE DI I/O (PERIFERICHE).

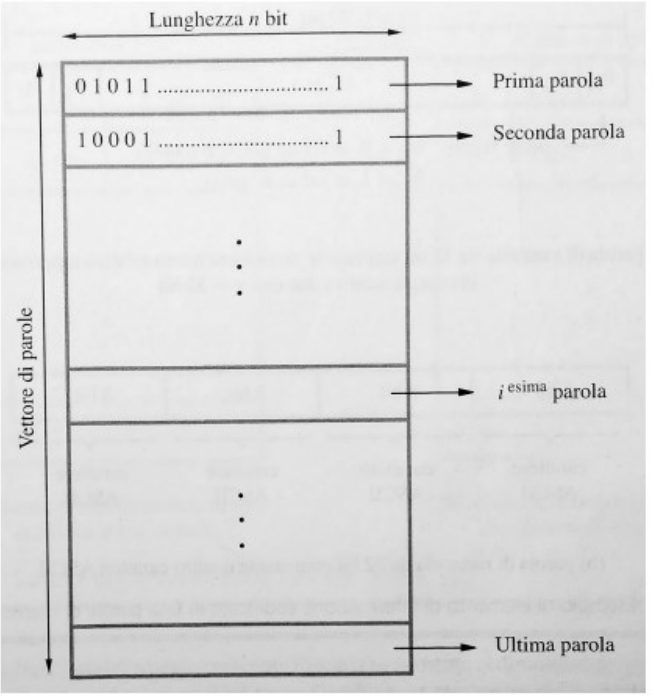
Come si programma?

- Linguaggi ad alto livello (C, C++, etc.) ancora più espressivi
- Il **COMPILATORE** traduce il codice ad alto livello in codice assemblativo
- Il programmatore scrive i programmi in **LINGUAGGIO ASSEMBLATIVO** (**ASSEMBLY**)
- Il programma assemblativo viene tradotto in **sequenze binarie** dall'**ASSEMBLATORE**



Memoria del calcolatore

- L'informazione è immagazzinata in memoria sotto forma di un vettore di **parole** (parole in successione), ognuna di **lunghezza** (o dimensione) n che varia da 16 a 64 bit.
- Ad ogni parola nel vettore è **associato un indirizzo binario** univoco. Per convenzione l'indirizzo è un numero naturale (sempre positivo o al più nullo) e convenzionalmente si usano gli indirizzi compresi nell'intervallo $[0,2^m)$ per un valore fissato $m \geq 1$, con m pari al numero di bit.
- La quantità totale di parole che la memoria contiene si chiama **spazio di indirizzamento**. Ad esempio con m = 24 per l'indirizzo si ha uno spazio di indirizzamento pari a $2^{24\ \text{parole}}$ di memoria.



Rappresentazione di istruzioni e dati

Le istruzioni e i dati sono rappresentati da SEQUENZE di CIFRE BINARIE (bit)

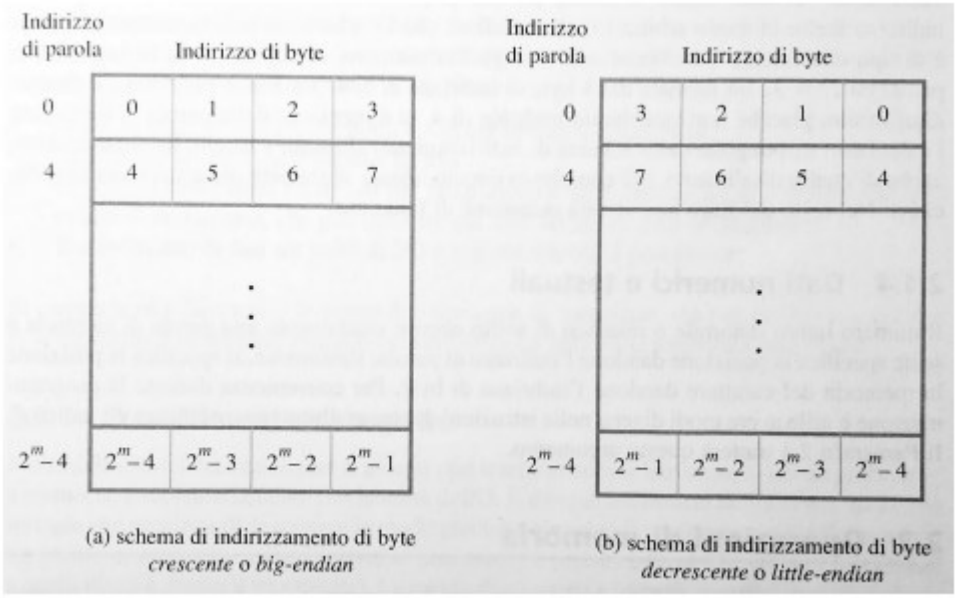
- Per convenzione una sequenza di **8 bit** è detta **Byte**
- I byte vengono raggruppati in blocchi con un numero di elementi espresso con potenze di 2:
- Kilobyte = KB = $2^{10} = 1024 \approx 10^3$
- Megabyte = MB = $2^{20} = 1024 * 1024 \approx 10^6$
- Gigabyte= GB = $2^{30} = 1024 * 1024 * 1024 \approx 10^9$
- Terabyte = TB = $2^{40} = 1024 * 1024 * 1024 * 1024 \approx 10^{12}$

Indirizzamento e ordinamento di byte

Di norma l'unità minima di informazione indirizzabile in memoria è il byte (ovvero 8 bit). Si assegnano indirizzi consecutivi ai byte contenuti in ciascuna parola. Gli indirizzi delle parole saranno quindi multipli della loro lunghezza in byte.

Vi sono 2 schemi di indirizzamento di byte:

- Crescente (big-endian): indirizzo aumenta **al diminuire** del peso aritmetico del byte
- Decrescente (little-endian): indirizzo aumenta **all'aumentare** del peso aritmetico del byte



(a) schema di indirizzamento di byte *crescente* o *big-endian*

(b) schema di indirizzamento di byte *decrescente* o *little-endian*

Insieme di istruzioni RISC e CISC

Reduced Instruction Set Computer (**RISC**):

- Insieme di istruzioni base ridotto.
- Ogni istruzione occupa una sola parola di memoria.

Complex Instruction Set Computer (**CISC**):

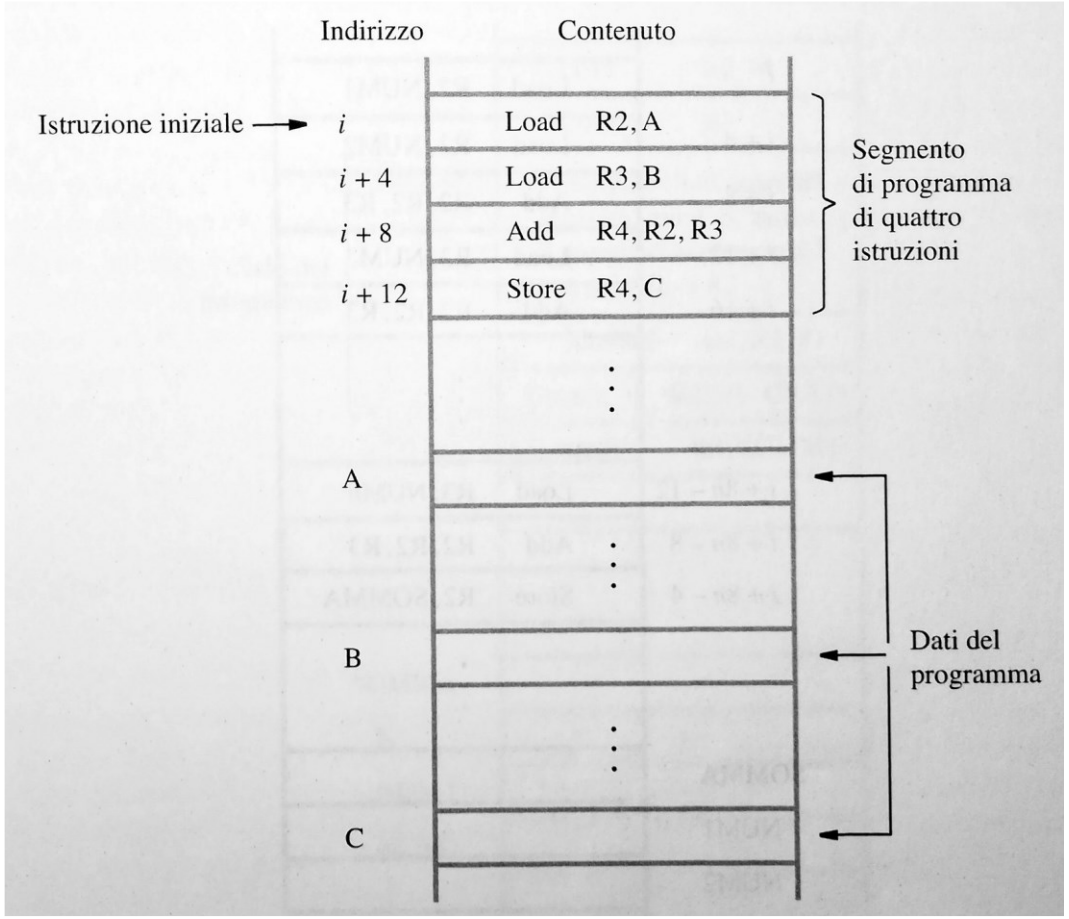
- Insieme di istruzioni base complesse.
- Ogni istruzione può occupare più di una parola di memoria.

Lista Istruzioni

Load	**R2, LOC**	Legge i contenuti di una locazione di memoria il cui indirizzo è rappresentato simbolicamente dall'etichetta LOC e carica i contenuti nel registro R2 del processore. I contenuti originali della locazione LOC sono preservati mentre quelli del registro R2 sono sovrascritti.
Add	**R4,R2,R3**	Somma i contenuti dei registri R2 e R3 e inserisce la loro somma nel registro R4. Gli operandi in R2 e R3 non sono alterati, ma il precedente valore in R4 è sovrascritto dalla somma.
Store	***R4,LOC***	Copia l'operando del registro R4 nella locazione di memoria LOC. I contenuti originali della locazione LOC sono sovrascritti mentre quelli del registro R4 sono preservati.
Move	**R4, R3**	Carica il contenuto del registro R3 in R4
Subtract	**R4,R2,R3**	Sottrae i contenuti dei registri R2 e R3 e inserisce il risultato nel registro R4.

Esempio di programma di somma

- Semplice programma che somma due valori presenti in memoria e ne salva il risultato.
  - **C ← [A] + [B]**
- Programma composto da 4 istruzioni (2 Load, 1 Add e 1 Store)
- Le quattro istruzioni sono memorizzate in parole di memoria consecutive
- Istruzioni lette sequenzialmente

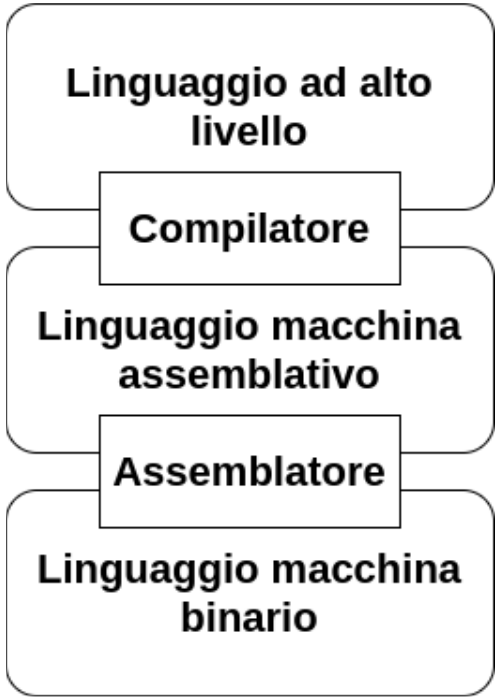


Modi di indirizzamento

I vari metodi mediante cui nell'istruzione macchina si può specificare la posizione di operandi e risultato si chiamano **metodi di indirizzamento**.

- Modo di registro
- Modo assoluto
- Modo immediato
- Modo indiretto
- Modo con indice e spiazzamento

Ogni istruzione contiene **codice operativo** (specifica l'operazione che deve essere eseguita) e **operando** (corrisponde ai dati su cui deve essere eseguita quella specifica operazione).



Modo di registro

L’operando o il risultato è contenuto *in un registro di processore*, il cui nome (che è indirizzo) è dato nell’istruzione.

L’istruzione

Add R4, R2, R3

usa il **modo di REGISTRO** per tutti e tre gli operandi. I registri R2 e R3 contengono i due operandi sorgenti, mentre R4 è la destinazione.

Modo assoluto (o diretto)

L'operando o il risultato è contenuto *in una parola di memoria* il cui indirizzo è dato nell'istruzione.
Il modo assoluto può rappresentare variabili globali in un programma.
Una dichiarazione del tipo:

Integer NUM1, NUM2, SOMMA

in un linguaggio ad alto livello farà in modo che il compilatore allochi una locazione di memoria per ognuna delle variabili NUM1, NUM2 e SOMMA. Quando successivamente si hanno riferimenti a esse nel programma, il compilatore può generare istruzioni assemblative che usano il modo ASSOLUTO per accedere a queste variabili.

Il modo assoluto è usato nell'istruzione:

Load R2, NUM1

che carica il valore alla locazione di memoria NUM1 nel registro R2.

Modo immediato

L’operando è dato esplicitamente *nell’istruzione*.

Si precede la costante dal simbolo cancelletto: **#valore**

Esempio in cui si aggiunge il valore 200 al contenuto di R6 e si pone il risultato in R4:

Add R4, R6, #200

Modo indiretto

Il nome di un registro di processore contenente *l'INDIRIZZO di memoria* dell'operando o del risultato è dato nell'istruzione.

Viene rappresentato con il nome del registro tra parentesi tonde **(·)**

L'istruzione:

Load R2, (R5)

Il registro R5 agisce come puntatore alla lista, si accede a un elemento della lista usando l'indirizzo nel registro R5. Si indica il modo indiretto tramite le parentesi tonde

Modo con indice e spiazzamento

L’indirizzo effettivo di operando o risultato e ottenuto *addizionando un valore costante* (**spiazzamento**) *al contenuto di un registro* (**indirizzo**)

Per indicare indice e spiazzamento si usa la scrittura **X(Ri)**, dove **X** e lo spiazzamento e **Ri** e il nome del registro contenente l’indirizzo

Utile nel gestire vettori o liste

L’istruzione

Load R2, 20(R5)

X = 20 e Ri = R5.

Se in R5 vi è il valore 1000, si preleva il contenuto della locazione 1020

VisUAL

A highly visual ARM emulator

VisUAL

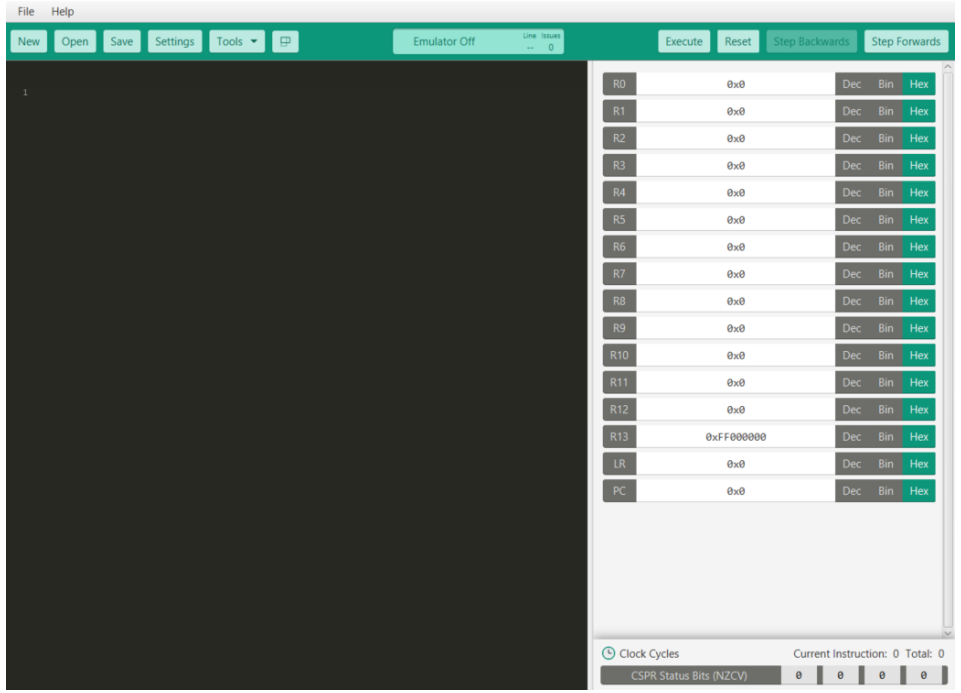
VisUAL

VisUAL è stato sviluppato come strumento multipiattaforma per facilitare l'apprendimento del linguaggio ARM Assembly.

https://github.com/tomcl/V2releases

GUI Visual

- **Registri di uso generale (R0,R1,..,R12)**: servono come deposito di dati di uso immediato e di puntatori a parole di memoria contenenti dati (o altri puntatori) su cui lavorare.
- **R13** (o SP) è generalmente il registro dello Stack Pointer, cioè l'indirizzo di memoria dello stack. Non è però obbligatorio il suo utilizzo come Stack pointer; viene usato solo per convenzione;
- **R14** (o LR) è il registro del Link Register, nel quale viene salvato l'indirizzo di ritorno nel momento in cui viene chiamata una procedura;
- **R15** (o PC) è il Program Counter, il registro contiene l'indirizzo di memoria della prossima istruzione da eseguire.



Modi di indirizzamento

- Registro: Ri
- Diretto (o assoluto): Locazione
- Indiretto: [Ri]
- Immediato: #Valore
- Base e spiazzamento: [Ri, #Valore]

Istruzioni

LDR Ri, [Rj]

Carica nel registro Ri il *contenuto della parola* di memoria puntata da R.

ADD Rd, Ri, Rj

Somma il contenuto di Ri e Rj e lo carica nel registro Rd

SUB Rd, Ri, Rj

Sottrae il contenuto di Rj da Ri e lo carica nel registro Rd

STR Ri, [Rj]

Salva nella parola di memoria puntata da Rj il contenuto di Ri

Altre Istruzioni

MOV Ri, Rj / MOV Ri, #Valore

Carica nel registro Ri il contenuto di Rj o il valore immediato

ADDS Rd, Ri, Rj

Somma il contenuto di Ri e Rj e lo carica nel registro Rd (**aggiorna** i bit di stato (NZCV))

SUBS Rd, Ri, Rj

Sottrae il contenuto di Rj da Ri e lo carica nel registro Rd (**aggiorna** i bit di stato (NZCV))

Direttive di assemblatore

Direttive di assemblatore

- L'assemblatore è in grado di tradurre un programma scritto in codice assemblativo (programma sorgente) in codice macchina binario (programma oggetto).

- Il linguaggio assemblativo non contiene solo le istruzioni del programma, ma anche comandi specifici per l'assemblatore (detti **direttive di assemblatore**)

- ***Le direttive di assemblatore servono per dare informazioni utili all'assemblatore.***

Dichiarazione di Eguaglianza

- Serve per associare un valore numerico ad un nome usato nel programma sorgente.
- ***SINTASSI:***
  - *NOME* ***EQU*** *valore_numerico*
- Per produrre il programma oggetto, l'assemblatore sostituirà ogni occorrenza della stringa *NOME* nel programma sorgente con il valore *valore_numerico*
- ***In VisUAL: EQU***

Dichiarazione ORIGIN

- Indica all'assemblatore l'indirizzo di partenza dove inserire le istruzioni e i dati definiti nelle righe seguenti
- ***SINTASSI:***
  - **ORIGIN** *Indirizzo_di_memoria*
- Alle istruzioni e ai dati seguenti la direttiva ORIGIN verranno assegnati gli indirizzi a partire dall'indirizzo *Indirizzo_di_memoria*

Dichiarazione RESERVE

- Indica all'assemblatore di riservare uno spazio di memoria espresso in byte
- ***SINTASSI:***
  - **RESERVE** spazio_in_byte
- La locazione di memoria riservata non viene inizializzata
  - ***In VisUAL: FILL***

Dichiarazione DATAWORD

- Indica all'assemblatore di riservare una parola di memoria e le assegna un contenuto
- ***SINTASSI:***
  - **DATAWORD** *contenuto_da_assegnare*
- La parola di memoria viene inizializzata con il valore *Contenuto_da_assegnare*
  - ***In VisUAL: DCD***

Dichiarazione END

- Indica all’assemblatore la fine del testo del programma sorgente
- ***SINTASSI:***
  - **END**
  - ***In VisUAL: END***

Linea di codice assemblativo

- In genere, una linea di codice assemblativo presenta i seguenti campi:
  - ***Etichetta*** ***Operazione*** ***Operandi*** ***Commento***

- **Etichetta:** Nome che viene associato all'indirizzo della parola di memoria assegnata all'istruzione o all'indirizzo del blocco di memoria riservato. E' facoltativa
- **Operazione:** Il nome (codice operativo) dell'istruzione oppure una direttiva di assemblatore.
- **Operandi:** Informazione di indirizzamento per accedere agli operandi
- **Commento:** Testo di commento, ignorato dall'assemblatore (marcato dai caratteri ";","%","!", o simili)

Esercizi

1) Sommare i numeri 2,3,4,1 e salvare in memoria il risultato

2) Effettua le due operazioni: 9 + 1 e 9 – 1 e salvare in memoria il risultato